CROSSFLOW-PLUS: AN SDN-ENABLED CROSS-LAYER ARCHITECTURE FOR WIRELESS NETWORKS

A Thesis

by

PRIYA MADHU SUNDARARAJU

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Alex Sprintson
Committee Members,	Srinivas Shakkottai
	Radu Stoleru
Head of Department,	Miroslav Begovic

December 2018

Major Subject: Computer Engineering

ABSTRACT

To meet the ever-growing network traffic demand, the underlying communication networks need to be open and programmable. The Software Defined Networking (SDN) paradigm separates the traditional data plane and network control plane, providing the ability to design, develop and manage the communication networks in an efficient and scalable manner. SDN enables a high degree of network programmability by providing high-level abstractions of the network devices and interfaces to manipulate them. SDN has attracted significant attention in the research community that produced a large body of work on programmable data plane and control frameworks. However, the existing studies mainly focus on the wireline networks, while wireless networks have received only limited attention.

Wireless traffic has significantly increased during the last couple of decades. Expansion of wireless communication networks is setback by limited usable spectrum, varying channel conditions, and signal fading. Envisioning to support re-configuration of wireless communication networks, fast dynamic reconfiguration of radio devices, and robust network programmability is necessitated. While network programmability is motivated by the SDN paradigm, Software Defined Radio provides a platform to perform various signal processing functions in software rather than hardware. However, to the best of our knowledge, there does not exist a framework to dynamically program wireless devices across all levels of the protocol stack.

Accordingly, in this project, we build the foundations of a fully programmable SDN-enabled framework that can achieve this goal. The proposed framework leverages the existing SDN approaches for programmable data planes as well as the Software-Defined Radio paradigm. In particular, we propose several key extensions to the SDN packet processing pipeline that enable different per-flow behaviors at the physical layer. The contributions of this thesis include: (i) a set of different physical layer profiles are presented as abstractions to the SDN application developer which is used to set different Physical layer parameters for different traffic flow type; (ii) extensions of SDN protocols (such as OpenFlow) to provide an abstraction for various physical layer profiles;

(iii) a proof-of-concept implementation and a set of use cases that demonstrate the benefits of our approach. We also perform an extensive experimental study to evaluate the performance gains that can be achieved by using fully programmable wireless devices.

DEDICATION

To my family and friends.

ACKNOWLEDGMENTS

I would like to express my utmost gratitude to Dr. Alex Sprintson for his continuous support and guidance throughout my graduate program. He provided encouragement and motivated innovative ideas towards solving different issues which led to this thesis work. I would like to thank Prithviraj Shome, author of a previous work, Crossflow for this technical help towards CrossFlow-PLUS. I would like to thank BAM! Wireless Team from Purdue University for their support and valuable input that provided direction to this work. I would like to thank Dr. Abhay Samant and Dr. Heena Rathore from National Instruments for their technical suggestions as well. A special mention goes out to Dr. Nicholas Mastronarde and Dr. Jalil Modares from the University of Buffalo for their constant support and valuable input towards Crossflow-PLUS. Finally, I want to thank Dr. Srinivas Shakkottai and Dr. Radu Stoleru for having served as my committee members.

Last but not the least, I would like to thank my family and friends for their unconditional support and immense encouragement.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Alex Sprintson, and Professor Srinivas Shakkottai of the Department of Electrical and Computer Engineering and Professor Radu Stoleru of Computer Science Engineering.

Help in design for test setup was provided by Dr. Jalil Modares and by Professor Dr. Nick Mastronarde of Buffalo University.

All other work conducted for the thesis was completed by the student.

Funding Sources

Graduate study was supported by a department scholarship from ECE department, Texas A&M University. This work is based upon work supported by the National Science Foundation under grant number 1642983.

NOMENCLATURE

SDN	Software Defined Networking
SDR	Software Defined Radio
UDP	User Datagram Packet

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGMENTS v

CONTRIBUTORS AND FUNDING SOURCES vi

NOMENCLATURE vii

TABLE OF CONTENTS viii

LIST OF FIGURES x

LIST OF TABLES xi

1. INTRODUCTION 1

2. BACKGROUND 4

    2.1 Software Defined Networking 4
    2.2 OpenFlow 4
    2.3 Software Defined Radios 5
    2.4 GNU Radio Framework 6

3. RELATED WORK 8

4. DATA PLANE EXTENSIONS 10

    4.1 Virtual Radio Ports in SDN switches 13
    4.2 Profiles in SDR domain 14

5. IMPLEMENTATION 15

6. TRAFFIC MODEL DESCRIPTIONS AND THEIR QoEs 19

    6.1 Web Browsing (HTTP) Traffic 19
    6.2 Voice-over-IP (VoIP) Traffic 20

7. VALIDATION 22

8. SUMMARY AND CONCLUSIONS 26

8.1 Challenges 26
8.2 Further Study 26

REFERENCES 28

APPENDIX A. MODULATION SCHEMES 31

A.1 Phase Shift Keying 31
A.1.1 Binary Phase Shift Keying 31
A.1.2 Quadrature Phase Shift Keying 32

LIST OF FIGURES

FIGURE Page

2.1 SDN Framework 5

2.2 OpenFlow protocol [Reprinted from flowgrammable [5]] 6

2.3 Architecture of a Software Defined Radio 7

4.1 UML diagram of OpenFlow data model depicting the relationship amongst elements Reprinted from flowgrammable [5] 10

4.2 Abstraction Interface 11

4.3 Data plane Pipeline 12

4.4 UML diagram for data plane model with modulators as introduced extensions 13

5.1 Radio Setup transmitting and receiving different traffic 15

5.2 CrossFlow-Plus Architecture 16

6.1 Web traffic 19

6.2 VoIP traffic 21

7.1 Setup for Per-packet Behavior SDN Application 22

7.2 Performance evaluation for different applications 24

A.1 Constellation plot for BPSK 32

A.2 Constellation plot for QPSK 32

LIST OF TABLES

TABLE Page

7.1 QoE Values for different Traffic Type. 23

1. INTRODUCTION

Software defined networking (SDN) paradigm has received significant attention from the research community and industry. SDN has emerged as a new and exciting tool for building flexible and efficient communication networks. The main principle of the SDN paradigm is decoupling of the control plane from the data plane. This decoupling enables the network operators to manage network resources in a highly adaptive way by re-configuring the network devices in the network on demand. In particular, the SDN concept provides an abstraction of the data plane and an Application Programming Interface (API) for SDN applications to interact with these abstractions. SDN applications are presented with the ability to run custom algorithms to control the data flows with a centralized view of the entire network. Network operators can develop SDN applications to implement their desired algorithms on the network elements.

Although SDN concepts for wired networks have been widely studied and researched, applying SDN principles for wireless networks has received significantly less attention from researchers and practitioners. Indeed, OpenFlow, currently most popular SDN southbound API, does not provide wireless abstraction. With the ever increasing demand of wireless traffic, scarcity of wireless spectrum, and constantly emerging new wireless protocols, there is a need to develop SDN approaches for wireless networks that would allow standardized on-demand reconfiguration of wireless devices and enable a high degree of programmability at the physical layer.

Software defined radio technology enables flexibility, cost efficiency and power to modify and reconfigure the radio devices. It provides the ability to design and implement almost all of physical layer operating functions, such as filters, amplifiers, modulators/demodulators, encoders/decoders, local oscillators, etc., through a modifiable software. This enables high flexibility in employing a general purpose radio hardware, connected to an RF front end, to implement different wireless services and protocols (e.g., GSM, Bluetooth, LTE, WiFi) with little or no additional hardware. While the SDR paradigm has revolutionized the digital signal processing in individual radios, to the best of our knowledge, it does not provide a defined method, with layers of abstractions, to

program a network of radios in an adaptive way.

Our aim is to provide a cross-layer architecture that has the ability to configure a broad range of network and physical layer parameters for different flows and modify these parameters in response to changing network conditions in a principled manner. Our dynamically programmable framework enables the network operator to:

1. *ensure that the available spectrum is utilized efficiently* by modifying the behavior of the network and/or physical layer parameters in response to congestion, user demands, and spectrum availability;

2. *prioritize flows* by differentiating between different data flows and provide preferential treatment of flows with stringent Quality of Experience (QoE) constraints and by selecting appropriate physical layer parameters e.g., selecting highly reliable coding scheme or higher transmit power for high-priority flows;

3. *gain logically-centralized knowledge* about the status of network links and elements and use this knowledge for optimizing network behavior;

4. *implement desired network policies* and algorithms through cross-layer control.

In this thesis we build the foundations of a fully programmable SDN-enabled framework for wireless networks. Our framework leverages both the recently developed SDN protocols (OpenFlow) [1] and the software defined radios (SDR) paradigm. OpenFlow is one of the most principled SDN frameworks that gained remarkable traction from the networking industry. Our framework extends the SDR functionality in a significant way that will allow the control plane to manipulate the important parameters at the physical layer and to fully integrate the radios into the broader SDN framework.

Our framework leverages the previous work, Crossflow Architecture presented in [18] which was the first attempt to define wireless abstractions for SDN applications. The main limitation of the Crossflow approach is lack of ability to support different per-packet behaviors. In contrast, our

architecture enables the network operator to handle packets of various applications, i.e., real-time data packets or HTTP packets in a different way, prioritize the desired packet flows, and to meet application-specific QoE requirements for a broad range of applications.

One of the main limitations of the existing SDN protocols, such as OpenFlow, is that they do not include abstractions for physical-layer components (such as modulators, coders, and transmitters). Accordingly, we define such abstractions as part of our framework and extend the OpenFlow protocol to provide a mechanism for the controller and the application developer to manipulate these abstractions.

To validate the approach, we implemented the architecture by extending the OpenFlow data plane pipeline to support wireless abstraction. We refer to our architecture as CrossFlow-Plus. Our approach provides a deep interaction between the different layers of the OSI model making it a true cross-layer architecture. Data plane of CPqD software switch is modified to support our architectural requirements. One is the main components of our architecture is the GNU radio framework that provides an open source digital signal processing platform for software defined radios. Several signal processing flows are designed using GNU Radio domain to implement and validate our architecture. GNU Radio interacts with the Software defined radios using UHD drivers to transmit and receive wireless data. A proof-of-concept SDN application, illustrating the per-packet behavior support, is designed and implemented using an open source SDN Controller, Ryu [2].

Our contributions can be summarized as follows:

- Designed a true cross layer architecture integrating SDN and SDR concepts;
- Implemented extensions to the Openflow data plane pipeline to handle wireless packets and connect to the software defined radios;
- Designed different signal processing flows on GNU Radio to handle and forward packets from the SDN data plane pipeline;
- Created an SDN Application depicting the per-packet behavior of our architecture.

2. BACKGROUND

2.1 Software Defined Networking

SDN paradigm can be defined as abstractions of basic data plane primitives (such as flow tables, queues, ports, and meters) and provide the interface that will allow the control plane to manipulate this abstraction. This interface is referred to as the Southbound API. Examples of such API include OpenFlow, Netconf, and Cisco OpFlex. The southbound API connects the individual switches with the controller. The key element of the control plane is a controller or, for distributed control plane, a set of controllers. The controller uses a south-bound API like OpenFlow protocol to dynamically add, delete or modify the flow table entries to perform certain packet processing actions. More generally, the south-bound API allows the controller to interact with data plane primitives, which includes receiving information about their capabilities, changing their configurations, obtaining statistics, and receiving call back information about the events, etc. The control plane enables development of the network applications that work with an abstracted view of the network to implement the policies and algorithms defined by the operator. SDN applications interact with the SDN controller via a north-bound interface. The high level scheme of the SDN framework is provided in Figure 2.1.

2.2 OpenFlow

OpenFlow protocol [1] is a southbound API which is designed to communicate with the OpenFlow switch using an SDN Controller. It provides an TCP/TLS connection to the controller making the connection secure between the switch and the controller. OpenFlow protocol provides a separation of the control plane from the data plane and a medium to interact between them using flow table entries. These flow table entries are composed of a number of packet details to match on, and provide actions on the packet depending on the matches. It can forward, add, modify or drop the packet depending on the actions in the flow table entry. OpenFlow protocol provides a communication medium to the controller to be able to add, delete, modify or forward the packets

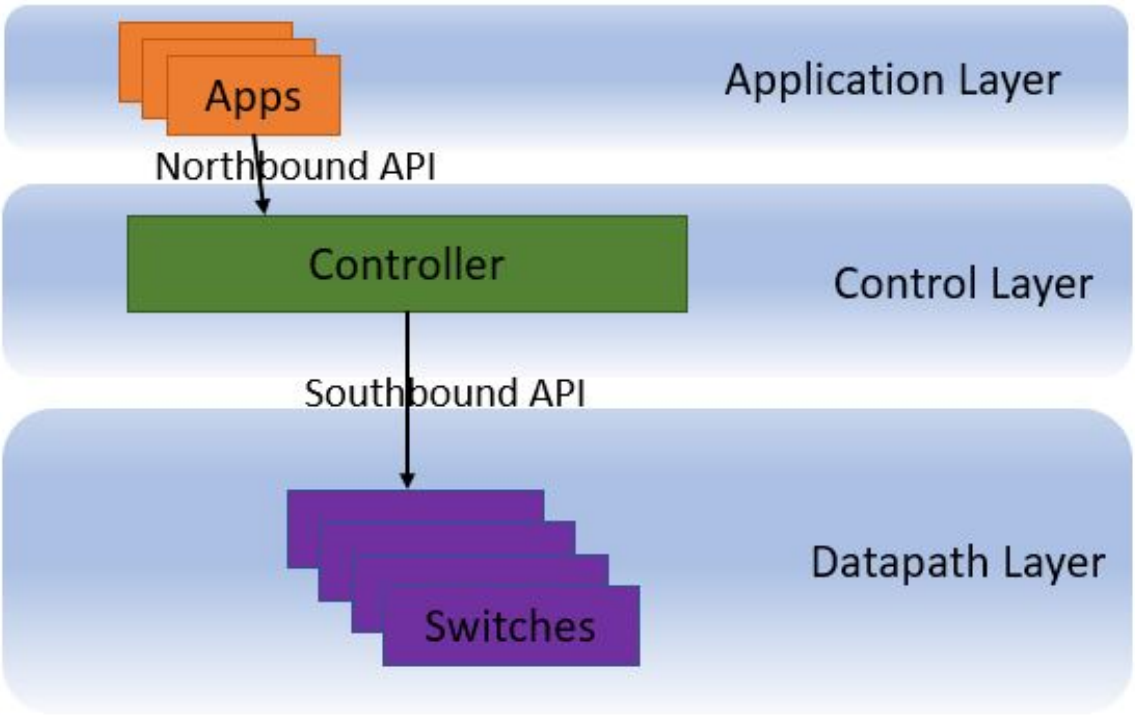

Figure 2.1: SDN Framework

according to the logical abstraction. OpenFlow switch agent is incorporated in SDN switches to process information received from the controller and update flow table entries. This provides the SDN controller with abstractions, hiding the underlying lower layer details. We use these SDN abstractions to change the physical layer parameters of wireless packets as well. We extend the SDN switch to be able to handle wireless packets using software defined radios and provide these extensions as abstractions to the SDN applications.

2.3 Software Defined Radios

SDR architecture can be viewed in terms of hardware and software. The radio consists of analog to digital converter/ digital to analog converter. These converters mark the boundary of software blocks to hardware blocks in a software defined radio. The hardware components are used to process the analog signals, while digital signals are processed in software. The ADC/DAC separates analog signal and digital signal processing from one another. This is separation is named

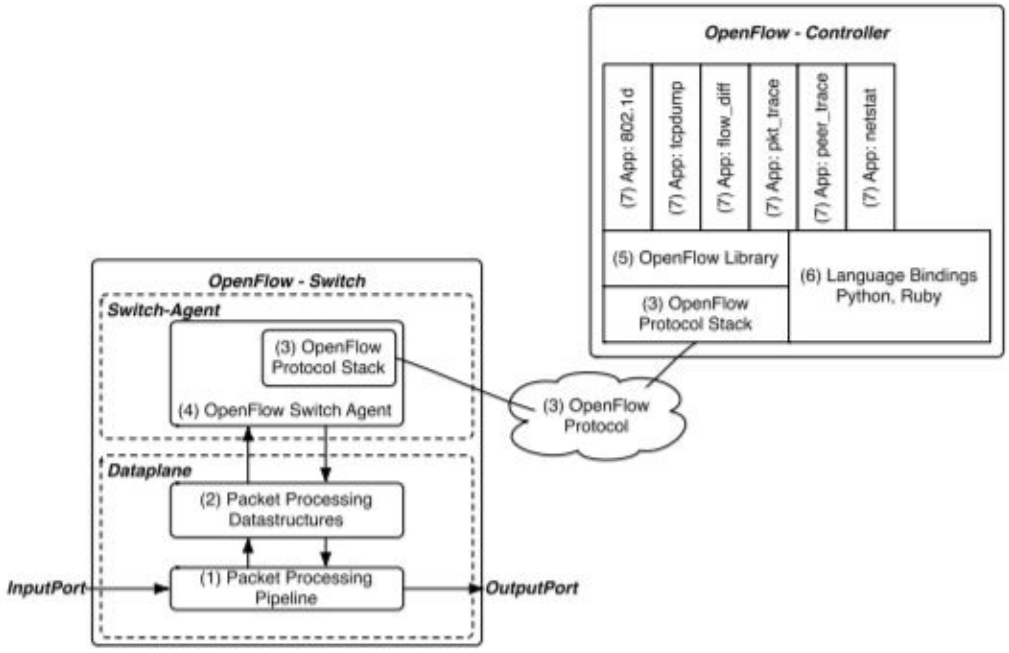

Figure 2.2: OpenFlow protocol [Reprinted from flowgrammable [5]]

as digital access point. All radio functions after the digital access point in the architecture are processed through software.

Software defined radios support a wide band of frequencies and dedicate the signal processing of digital data to the software blocks as shown in Figure 2.3. These blocks are put together to form a flow graph that provides digital data to the RF front end of the radio with the help of specific drivers. These drivers also convert the received analog signals into readable digitized form for the software (GNU Radio) domain to further process the received signals.

2.4 GNU Radio Framework

GNU Radio [3] is a free and open-source software development framework that provides signal processing support for software defined radios. The blocks provided by GNU Radio help perform distinct signal processing functions. These blocks are used to implement any radio functionality on the SDRs. However, the framework does not have the ability to provide the application developer

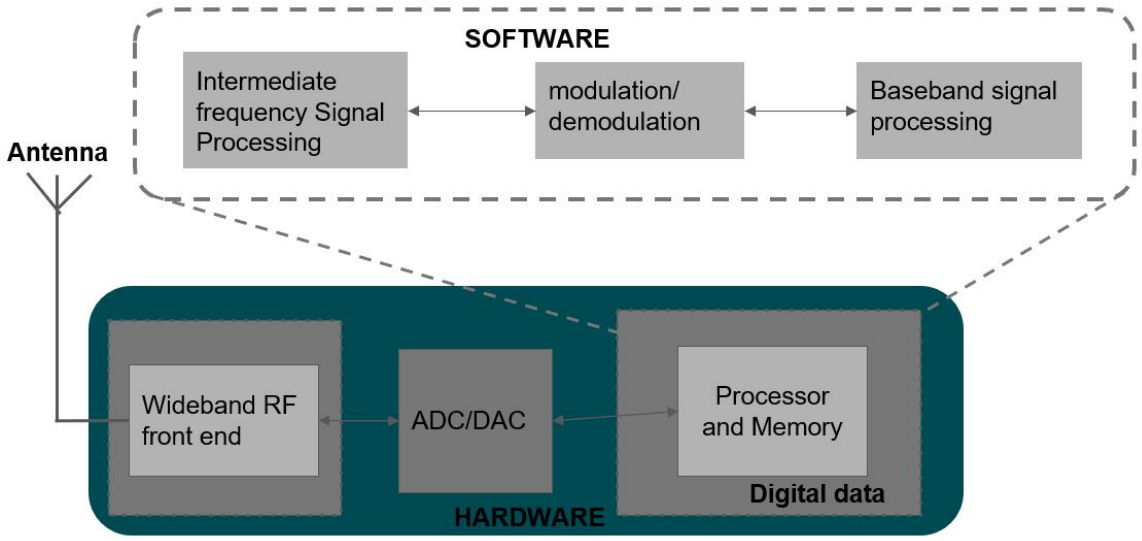

Figure 2.3: Architecture of a Software Defined Radio

with access to choose different blocks and control the network of SDRs. In this thesis, we provide the ability to expose this re-configurability of radios to the network operator and design the selection of blocks to effectively optimize the spectrum usage. It aids to meet different performance requirements for different types of packet flows.

3. RELATED WORK

The interest in software defined wireless networks and the integration of SDN and SDR concepts has attracted a large body of research over the last decade. An early work proposing a cross-layer architecture with integrated SDN and SDR is seen in Openroads [20]. Openroads presents an architecture that provides a support for program control in WiFi and WiMAX networks. It allows the network operator to design control algorithms for network slicing in cellular networks. An architecture similar to our proposed architecture is presented in Ætherflow [19] that provides a set of wireless abstraction for WiFi networks. However, these architectures are confined to WiFi networks. Also, proposed work in Programming Abstractions for Software-Defined Wireless Networks [17] provides a Python-based Software Development Kit to implement the proposed similar SDN abstractions for wireless network. However, these proposed abstractions are limited to WiFi networks whereas our work provides protocol-independent wireless abstractions.

OpenRadio [6] presents an cross-layer architecture for a programmable wireless data plane. The architecture provides flexibility to modify PHY and MAC layer. However, this flexibility is not extended to the Network or transportation layer. Similarly, in [7], programmable wireless data plane has been proposed. It provides modular blocks and focuses on processing signals. However, it does not provide a logical platform for network operator to vary PHY and MAC layer characteristics per flow type. In contrast, our work provides programmable access to design of signal processing on GNU Radio for USRPs which facilitates unlimited flexibility of PHY layer characteristics.

Integration of SDR and SDN for 5G [9], discusses the benefits on providing wireless abstractions for SDN application developers. It highlights the need for a cross-layer controller. However, it does not provides a working model of the proposed architecture. In our work, we build a working model of the architecture using SDN switches and SDR radios.

RCube [10] proposes a cross-layer architecture that provides a structured methodology to implement complex decision-making algorithm in modular and protocol independent manner. The

design is structured into decision, control, data and register plane. However, for different protocol, the network operator has to provide a module for packet abstraction and protocol implementation in the register plane. In our work, data plane pipeline of software defined network switch is embedded with protocol-independent packet abstractions. This pipeline does not require extra pipeline to be added for every varying protocol. Instead, flow rules are added to provide logical actions for different packet/protocol type.

CrossFlow [18] proposes an cross-layer architecture that implements integration of SDN and SDR paradigm to provide a platform for network operators to use wireless abstractions for reconfiguration of radios and network switches. However this architecture does not support per-packet behaviour for wireless abstractions that are defined in this work.

Architecture presented in softRan, [12] deals with centralized control of radio devices in a wireless network. It provides a logical abstraction to vary the transmit power and spectrum resource allocation for radio devices based on optimization algorithms. However, it does not provide ability to vary modulation schemes based on the network. Also, it focuses mainly on LTE network whereas, our work provides a mechanism for centralized control while making the exposed interfaces protocol independent.

Work in [13] describes a Software Defined Physical layer architecture which presents an integration of SDN and SDR paradigm. However, this architecture is specific to LTE Networks. Architecture presented in [14] provides a blueprint for LTE self-organizing networks (SONs) using SDN and SDR principles. These papers provide distinct solutions for various scenarios but do not provide a generic framework for handling various protocols and various traffic flow type in a principled manner. This is addressed in this thesis.

4. DATA PLANE EXTENSIONS

In TinyNBI [8], SDN abstractions model is derived from openFlow specifications [1]. Data plane elements are listed in [1], and the relationship between these elements in shown in Figure 4.1.

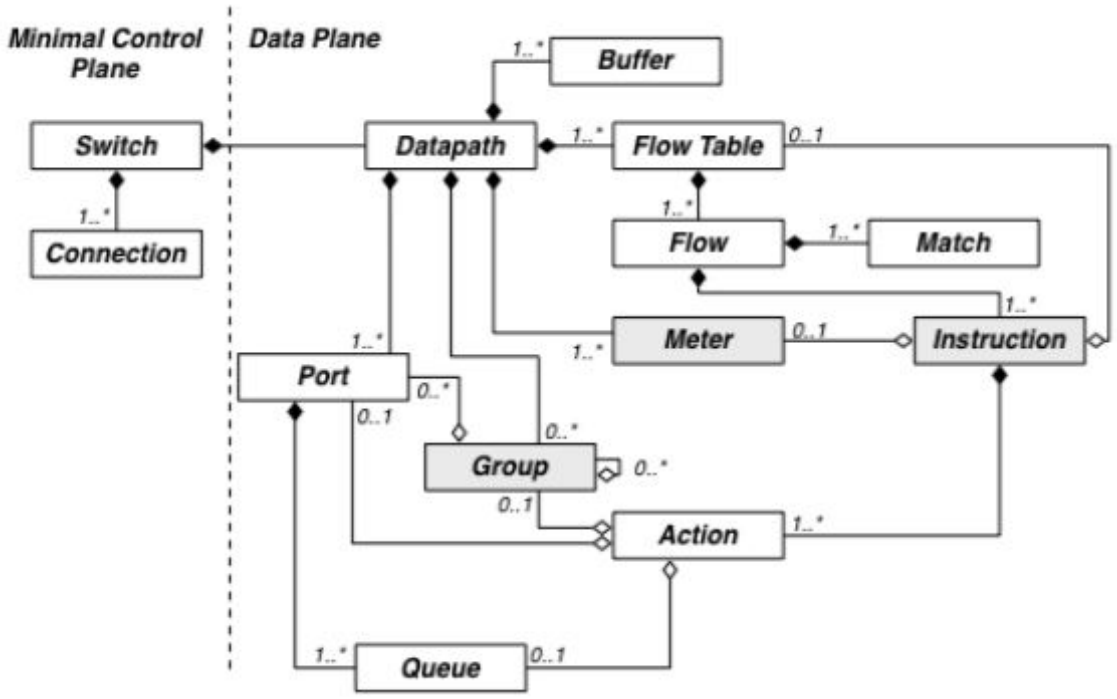

Figure 4.1: UML diagram of OpenFlow data model depicting the relationship amongst elements Reprinted from flowgrammable [5]

In the TinyNBI model, a generalized SDN abstraction model [8], each component exposes four types of interfaces: capabilities, configuration, statistics and events, as shown in Figure 4.2.

1. The capabilities interface allows the controller to receive information about the set of operations supported by the device. It is a read-only interface.

2. The configuration interface includes functions for all the configurable parameters that can be modified during switch operation.

3. The statistics interface provides the controller with average operational information such as average packet rate, drop rate, and more.

4. The events interface enables the controller to be notified through call-back calls when certain pre-specified conditions occur.

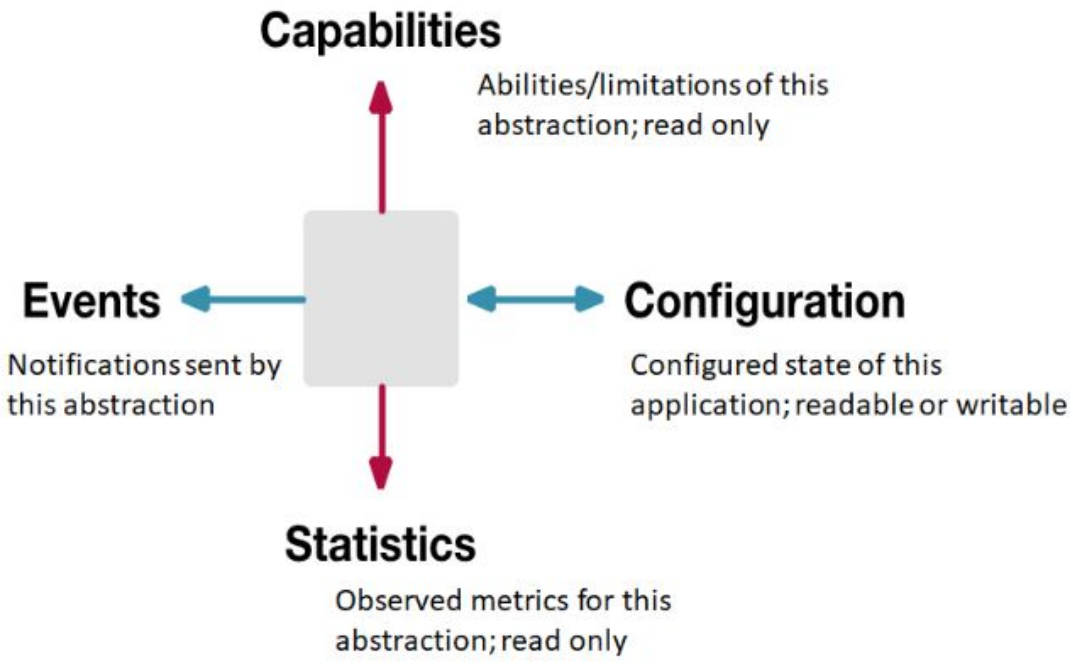

Figure 4.2: Abstraction Interface

The first step in enabling programmability is to define specific abstractions as described in Figure 4.2 and interfaces for the physical layer implemented by the software-defined radios. The previous work in this direction, CrossFlow [18], has defined a set of abstractions for the data plane, along with the interface to manipulate them. However, CrossFlow [18] did not provide the system with the ability to use different physical layer profiles for different packets. Accordingly, in this

work, we propose a new architecture that allows to define several physical layer profiles and to use appropriate profiles for packets that belong to different flows.

The proposed architecture enables coordination between various layers of the protocol stack and enables the network layer characteristics to define the physical layer parameters when the packet has been transmitted. This architecture provides for a deeper interconnection of and inter-dependency between different OSI layers, making our design a true cross-layer architecture.

SDN switches execute the data plane pipeline that provide modules for packet header extraction and a structured manner to manipulate packet content and forwarding. However, the present pipeline model is only able to process wired packets. Hence, our architecture uses and extends the data plane pipeline to enable processing of wireless packets.

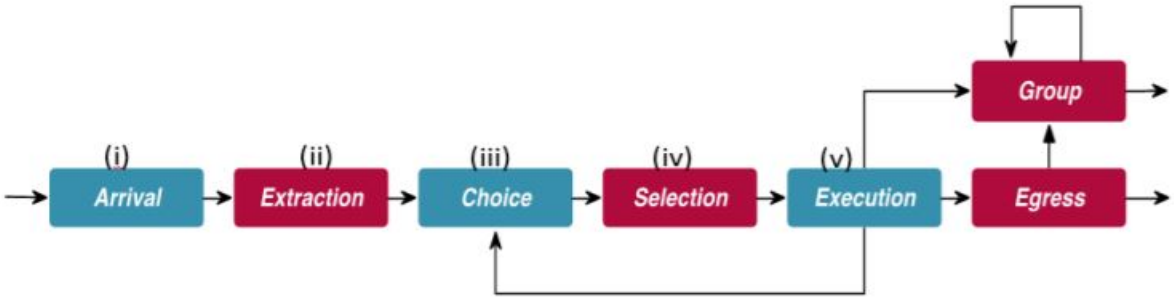

Figure 4.3: Data plane Pipeline

Figure 4.3 illustrates the steps involved in data plane pipeline of an SDN switch. The packets are processed in the following order: (i) the packet arrives at the network element, (ii) the headers are extracted to provide information about the packet, (iii) the appropriate flow table is found, (iv) the packet is matched against the flow entries. The flow entries determine the next course of the packet as they can (v) modify, delete or forward packets to different ports. Our work involves defining multiple virtual radio ports to which the wireless packets are forwarded using flow table entries.

4.1 Virtual Radio Ports in SDN switches

The virtual radio ports in SDN switches expose radio abstractions to the SDN control plane. They provide a mediation layer between the SDN switch and the SDR signal processing blocks. Once the data packet flows through the data plane pipeline, the packet is forwarded to an appropriate virtual port. The wireless data packets exiting the data plane pipeline are received by the virtual ports, formatted in order to pass through the SDR signal processing blocks and forwarded to the appropriate SDR signal processing thread. Note that a connection to the SDR domain from the SDN switch is provided by the implementation of virtual ports without changing the structural pipeline flow of the data plane. It is important to ensure minimum changes are made to the data processing pipeline to maintain its compatibility with the existing SDN pipelines employed by the wireline networks. The SDN control layer is provided with radio abstractions while the underlying physical layer complexity is effectively hidden by the instances of virtual radio ports. For example, in the UML diagram of OpenFlow data model is introduced with an added element, modulators, which connects to the virtual wireless port as shown in Figure 4.4.

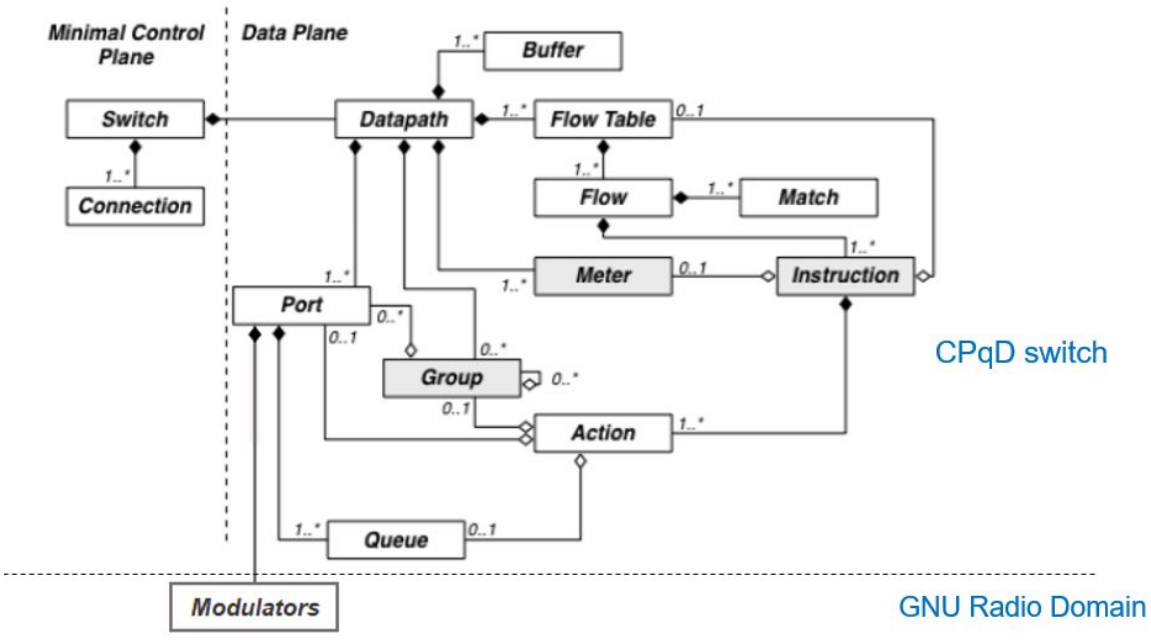

Figure 4.4: UML diagram for data plane model with modulators as introduced extensions

Also, the virtual port may be provided with events generated by different components of the physical layer signal processing pipeline, such as low signal-to-noise (SNR) ratio, low signal strength, etc.

4.2 Profiles in SDR domain

At the SDR domain, profiles are introduced for different radio parameters. For example, a set of profiles could be (i) high data rate (QPSK) with low transmit power (5 dB antenna gain) and (ii) low data rate (BPSK) with high power (15 dB antenna gain). Intuitively, profiles define different physical layer parameters and schemes that can be used for transmitting the packet. For example, the first profile can be used for low priority interactive data type in a congested environment, whereas the second profile can be used for highly error-sensitive data traffic type. Each profile has an independent signal processing flow in the SDR domain. These profiles, or multiple signal processing flows are implemented in our design to facilitate faster switching between radio parameters. In contrast, CrossFlow architecture supports only one profile, limiting its flexibility and treats all types of traffic packets, belonging to different flows, in the same way. Different profiles are mapped to the virtual ports on the SDN domain in our architecture. The wireless data packets are matched and forwarded to the appropriate virtual radio port on the SDN switch by the flow table pipeline and then is sent to the corresponding signal processing module in the Software Defined Radio domain. This enables the SDN application to define different behaviors for different types of packets.

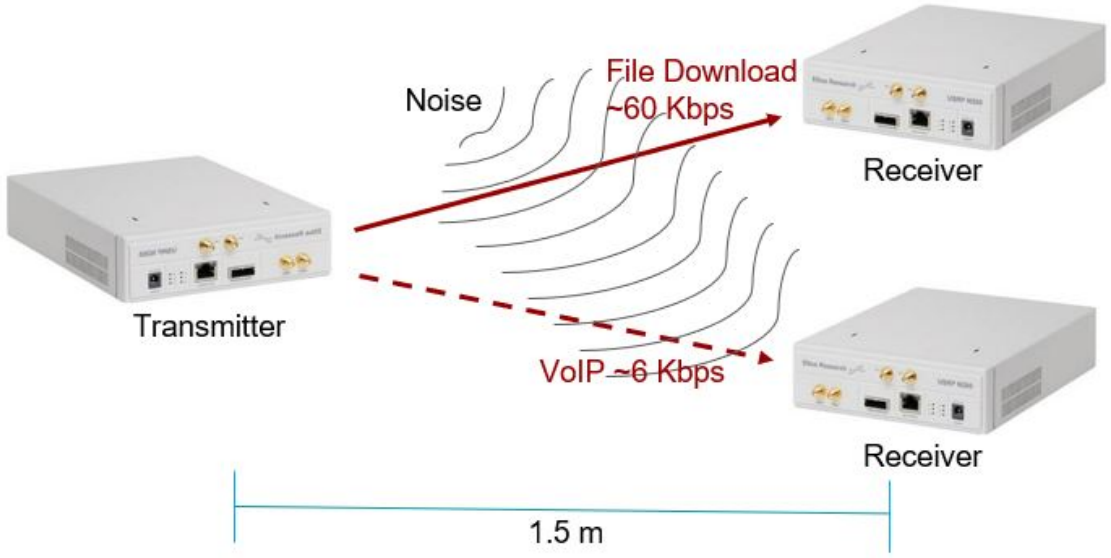

Figure 5.1: Radio Setup transmitting and receiving different traffic

For implementation and validation purposes, Universal Software Radio Peripheral (USRP) N210 embedded SDR from Ettus Research is used. The N210 provides a Zynq 7020 All Programmable SoC, which combines a dual ARM Cortex-A9 processor and FPGA on the same device. USRPs connect to a host computer (PC) through a high-speed link, 1 Gbps ethernet connection, that is used by the host-based software (GNU Radio) to control the USRP hardware which transmit/receive data over the air medium. GNU Radio [3], a project written in C++, provides a platform for signal processing in software and uses UHD drivers to connect to the USRP hardware. The different signal processing flows or profiles in the radio is written using GNU Radio software as well as the mediation layer, to SDN switch is supported by this software. For the switch agent in the SDN model, an existing implementation of SDN switch, CPqD switch is used. It is an open-source software that is modified to support virtual radio ports in our work. Ryu software [2], an open-

source SDN controller platform provides us with the Northbound API to write SDN applications on it. Various Southbound API protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc. is also supported by Ryu software. An SDN application for Ryu controller is written to illustrate the per-packet behavior on our architecture. The CPqD software switch implementation is modified to incorporate virtual radio ports. The required mediation layer between SDN switch and SDR domain is provided by these virtual ports.

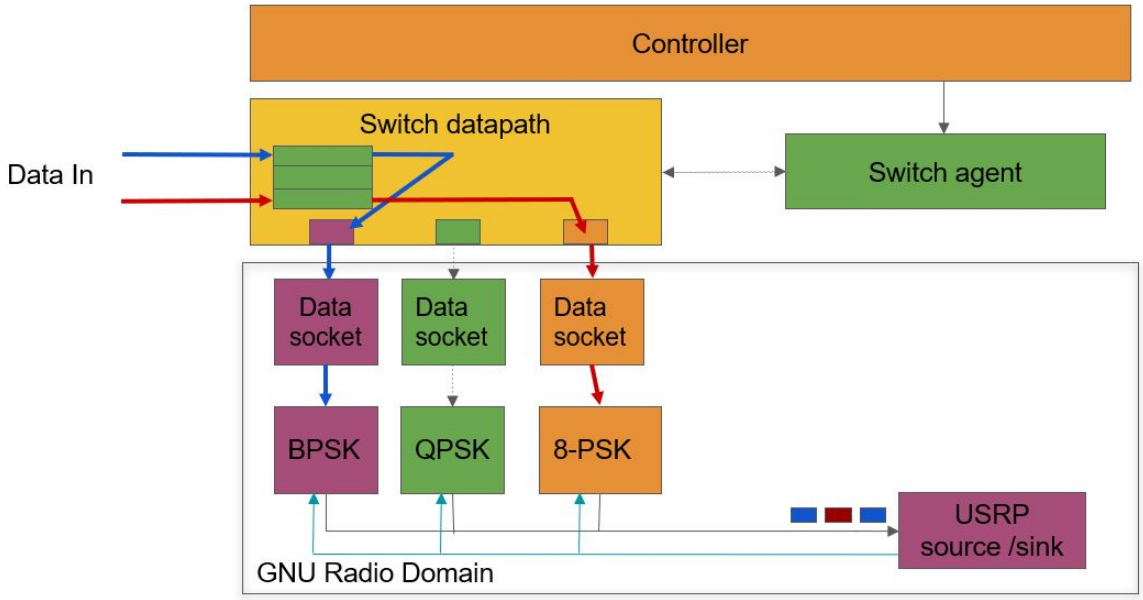

Figure 5.2: CrossFlow-Plus Architecture

Data packets are forwarded by the switch to the data socket present on the GNU Radio. Note, each virtual port is connected to a different data socket, providing the SDN application the ability to switch radio parameters, in our case, modulation schemes, as shown in Figure 5.2.

The major blocks used in implementing varying modulation signal processing pipeline are as follows:

- *Data socket* block connects various virtual ports in the CPqD switch to GNU Radio flow graphs over a TCP connection. An inter-process communication pipe is used to transfer data

packets to different PHY layer parameters, according to the SDN application rules.

- *Packet encoder* adds a block of access code as a simple MAC layer header, for packet retrieval from a string of bits.

- *Signal processing* for data packets occur after the packet is received from the switch and appended with an access code. The packets are tagged before they are converted to a stream of bits to mark the start of the packet. The tagged bits of data are then modulated with a specific modulation scheme.

- *Tagged Stream Multiplexing* custom block is used to select the flow that sends data to the RF sink. If there is presence of data in two flows with differently modulated bits, the multiplexer buffers one flow while outputs the other that use the RF resources to transmit packets over air.

- *UHD Sink* block provides the USRP with digital data that is to be transmitted and sets the center frequency and gain values of transmission.

The receiver architecture is very similar to the transmitter. The virtual ports pick up the packets coming from the radio and forward it according to the SDN application rules.

The major blocks used in receiving and constructing packets from signals are as follows:

- *UHD source* translates the radio signal into digital signals and provides it as input to the following signal processing blocks. It is set to the same center frequency as the transmit center frequency.

- These bits are now sent to different *signal processing* pipelines. All these signal processing pipelines have varying type of demodulation blocks. Once the signal is demodulated, it is passed on to the packet decoder block.

- *Packet decoder* block matches the received and demodulated bit sequence to the access code sequence that was appended as a header to the transmitted packets, marks the start of the packet and removes this header before passing it to the next block.

- *Data socket* transfers the data packet to the OpenFlow switch, which then flows through the data plane pipeline to reach its destination.

6. TRAFFIC MODEL DESCRIPTIONS AND THEIR QoEs

Two different type of network services, VoIP and Web browsing, are discussed in detail along with its user-oriented Quality of Experience(QoE). QoE according to [15] may be defined as "overall acceptability of an application or service perceived subjectively by the end-user". The relationship between network oriented Quality of service (QoS) and Quality of experience (QoE) is illustrated in [11]. The traffic model and QoE for VoIP and web-browsing services are discussed below.

6.1 Web Browsing (HTTP) Traffic

HTTP protocol is foundation of data communication for World Wide Web traffic. It commonly has a bursty profile due to its interactive nature, where the HTTP traffic pattern is as shown in Figure A.1.

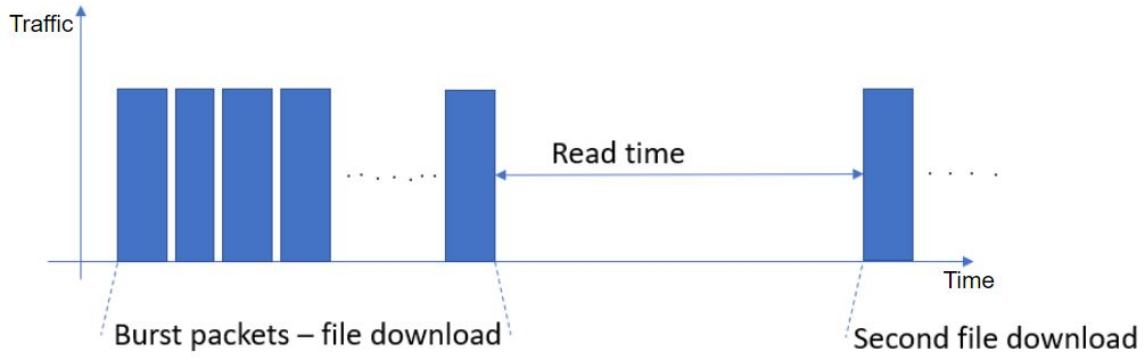

Figure 6.1: Web traffic

HTTP traffic is asymmetric, bi-directional. It has large, fixed-size block data frames sent from a HTTP server to a HTTP client, and HTTP request packets and TCP ACK responses received from the HTTP client to a HTTP server. However, only block of data frames from HTTP server to HTTP client is considered in our proof-of-concept HTTP application protocol.

The web session consisted of three steps, reflecting a typical search-for-information situation involving (a) requesting a search page; (b) typing and submitting a query; and (c) retrieving the results. According to [11], QoE is inversely exponential to session time. In our experiments, we consider page download or results retrieval time only. However, retrieval time contributes to the session time and hence, deduce QoE to be inversely exponential to retrieval time too.

t $\propto T$ where *t* is page retrieval time and *T* is the session time

According to [11],

$$QoE(T) = 1.390 + 4.298exp(-0.347T) \tag{6.1}$$

where *T* is the session time

We assume,

$$QoE(t) = 1.390 + 4.298exp(-0.347t) \tag{6.2}$$

where *t* is the page download or retrieval time

to evaluate QoE for http page download. We compare file download traffic to be similar to HTTP page download traffic model. Hence, we use equation 6.1 to evaluate QoE for file download traffic in our experiments.

6.2 Voice-over-IP (VoIP) Traffic

Voice over IP traffic is a best effort traffic without any QoS guarantees. We consider 6 Kbps data rate in our experiments. The Data rate of VoIP traffic is typically between 6 Kbps to 12.4 Kbps. It is symmetric, bi-directional between source and sink. Evidently, VoIP user will always be in silent state or active talking state.

For VoIP traffic, QoE is considered to be inversely exponential to packet loss rate. The quantification of the QoE is done using the Perceptual Evaluation of Speech Quality (PESQ) method described in ITU-T P.862[16]. The equation 6.3, depicts the relationship of QoE and packet loss ratio for VoIP traffic. According to [11], the QoE for VoIP traffic can be expressed as,

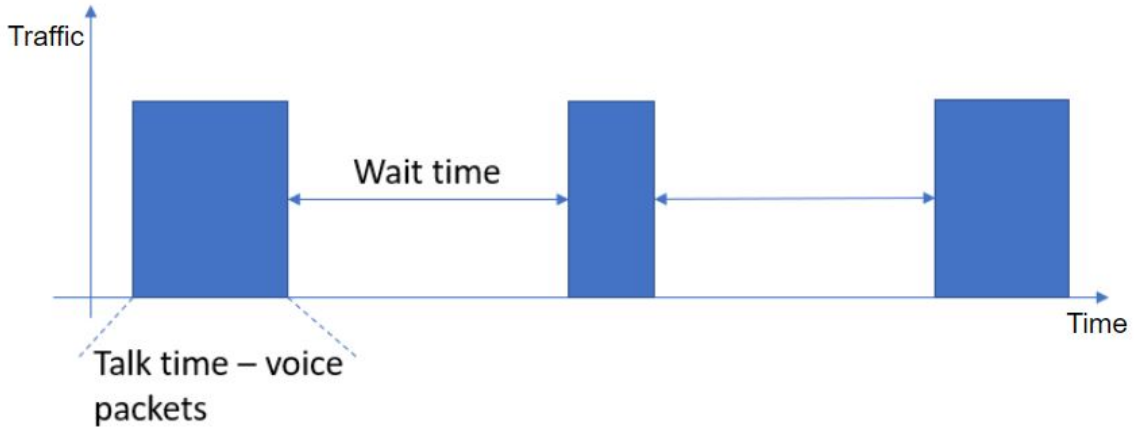

Figure 6.2: VoIP traffic

$$QoE(L) = 1.065 + 3.010exp(-4.473L) \tag{6.3}$$

where L is the packet loss rate

7. VALIDATION

To demonstrate that CrossFlow-PLUS architecture can dynamically reconfigure radio parameters, a simple wireless Qualify of Experience (QoE) application is designed and implemented. This application chooses appropriate modulation schemes, a radio abstraction, for different types of traffic flows. Our setup involves a source, that transmits file download packets and low data rate packets similar to real-time packets over UDP to sinks, as shown in Figure 6.1.

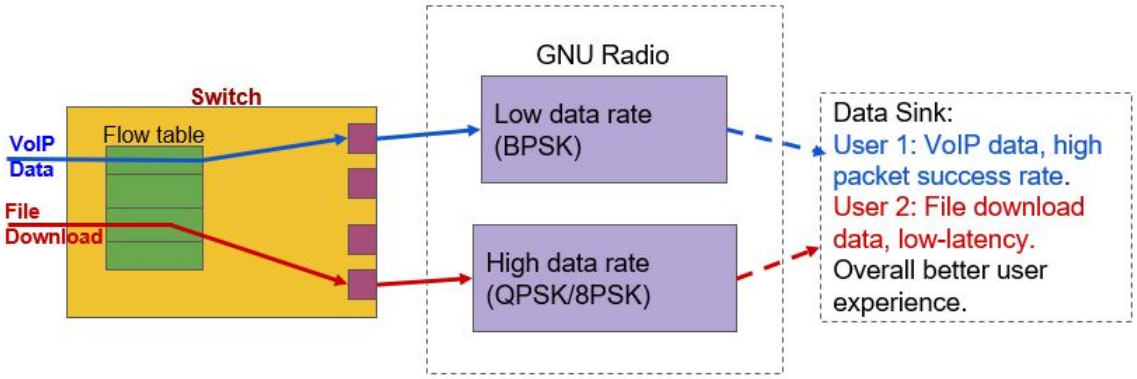

Figure 7.1: Setup for Per-packet Behavior SDN Application

The source and sinks for different data traffic are virtual hosts defined in the PC that runs the CPqD switch [4] and SDN controller. They are connected to the software switch. This virtual network is defined, using LINUX commands, to validate our Architectural design.

The USRP N210 device is used to transmit and receive the different traffic packets. It is connected to the CPqD ofsoftswitch running on the PC through the GNU Radio. The GNU Radio software runs on the same PC as well. The transmitter and receiver is set 1.5 meters apart. The sender radio transmits with 20dB antenna gain whereas the receiver is set to 30dB. The sender begins to transmit at 900 MHz carrier frequency with 30 KHz bandwidth. file download packets are 60 bytes long and 500 packets are transmitted for a single page download. VoIP packets are

transmitted with an interval of 60 ms. File download packets are transmitted with 60 Kbps whereas VoIP packets are transmitted at 30 Kbps.

Defined radio abstractions are used by SDN application to select high data rate modulation scheme for file download packets, whereas high reliable modulation scheme for VoIP traffic. The characteristics of different traffic types and their QoE values are explained in detail in Appendix [6]. As a result, the QoE of all traffic flows combined is improved. The radio abstractions used by the SDN application is provided by CrossFlow-PLUS architecture to optimize usage of wireless network resources and match the QoE requirements for different traffic flows. Without the SDN application policies and rules, all the file download packets and VoIP packets have the same radio parameters, for example, same modulation scheme. Also, previous work, Crossflow Architecture does not support per-packet behaviour. That is, equal number of packet-drops and data rates is experienced by all flows. This provides good QoE for one type of flow but the other type of traffic flow suffers.

When SDN application is implemented, (i) it matches the traffic type of the data packet to the SDN application rules, (ii) chooses high-reliable transmission profile for VoIP packets and high-speed transmission profile for file download packets, and (iii) forwards the packet to appropriate signal processing flows in SDR. As a result, the number of packets dropped in VoIP connection is lesser, since, the application chooses a more reliable connection for it. Also, the data rate of file download connection is higher, making it more suitable for faster reception and lesser spectrum usage. Hence, the end-to-end delay of both the flows is minimized. The use of virtual ports reduces the switching time, making it less expensive even in highly changing network environments.

The different averaged QoE values for different cases is tabulated in 7.1

Traffic Type	Only QPSK Mod	Only BPSK Mod	SDN Solution
File Download	2.420196	1.755435	2.41629
VoIP	1.520267	2.27272	2.30109

Table 7.1: QoE Values for different Traffic Type.

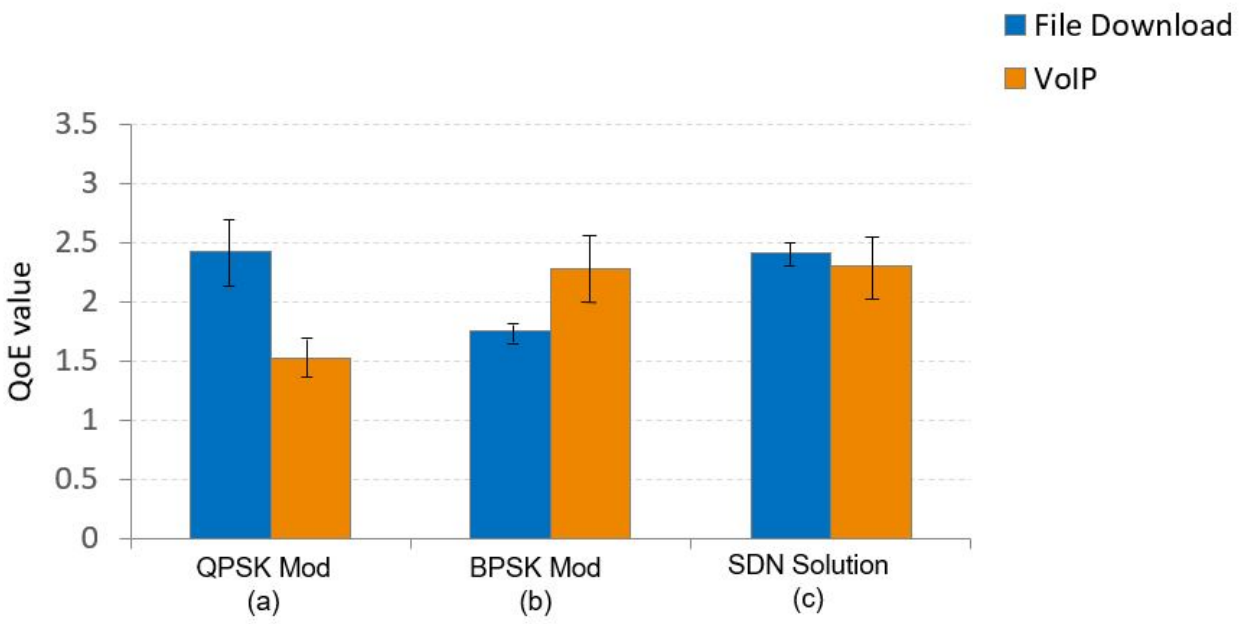

Figure 7.2: Performance evaluation for different applications

Figure 7.2 (a) depicts that the QoE value for file download is high, however QoE for VoIP is low, when both traffics are transmitted using QPSK Modulation. QPSK Modulation provides higher data rate compared to BPSK for the same bandwidth value, however it suffers a higher bit error rate. Due to the characteristics of QPSK Modulation, explained in detail Appendix [A], it is able to transmit the file download packets at a faster rate and get a better QoE value but VoIP packets suffer high packet loss rate and hence the QoE value is affected.

Also, 7.2 (b) illustrates that the QoE value for VoIP traffic is high, however QoE for file download is low, when both traffics are transmitted using BPSK Modulation. BPSK Modulation has lower data rates compared to QPSK for the same bandwidth value, however it is highly reliable with low data loss rate. Due to the characteristics of BPSK Modulation, explained in detail Appendix [A], it is able to transmit VoIP packets with less packet loss rate and hence, high QoE value, whereas the file download packets suffer a low QoE value due to the delayed reception of data packets.

Using the SDN solution, we select appropriate modulation schemes for appropriate traffic

flows. Figure 7.2 (c) depicts that selection of QPSK modulation for file download packets and BPSK modulation for VoIP packets, gives a better QoE value for both which averages out to 2.41629 and 2.30109 respectively.

8. SUMMARY AND CONCLUSIONS

In this paper a SDN framework, CrossFlow-PLUS was presented. This framework is designed to enable wireless abstractions for SDN application developers. It is a protocol independent architecture and provides programmability and flexibility of wireless networks to application developers for design and implementation of algorithms to render better user experience. It provides the ability to vary different lower layer parameters using network and transport layer parameters. Hence, it porvides an ability to vary PHY layer parameters for different types of traffic flows. CrossFlow-PLUS provides these abstractions by exposing virtual wireless ports to the SDN application developer but effectively hiding the underlying complex radio RF implementation. These abstractions are validated by running experiments with two traffic flow types and selecting appropriate modulation schemes for different traffic flows.

This validates that our design is able to influence physical layer parameters by matching network and transport layer characteristics and can be used to implement and test any algorithm designed with such requirements.

8.1 Challenges

There are multiple demodulation blocks demodulating the received signals with different modulation schemes, although only one block demodulates and retrieves the correct packet for the signals. Since, the receiver is unaware of the modulation scheme used for the packet, it demodulates on all available flows. This consumes large CPU resources, multiple times more than needed. Hence, the number of signal processing flows and the schemes to select from, is limited by the receiver′s CPU.

8.2 Further Study

The uses of multiple demodulating blocks can be avoided by implementing training sequence before transmission of data packets and enabling only one demodulating flow graph that receives the correct training sequence. This would require the controller to transmit training sequences

before transmission of data. The receiver decodes the control packets and selects on the correct demodulating flow.

REFERENCES

[1] Open Networking Foundation. Openflow-enabled mobile and wireless networks. Technical report, September 2013. https://www.opennetworking. org/images/stories/downloads/sdn-resources/solution-briefs/ sb-wireless-mobile.pdf [Online; Accessed: 2014-08-26].

[2] RYU Controller. http://osrg.github.io/ryu/. [Online; accessed 23-Sep-2015].

[3] GNU Radio. http://gnuradio.org/redmine/projects/gnuradio/wiki. [Online Accessed: 2015-09-30].

[4] CPqD OpenFlow 1.3 Software Switch. http://cpqd.github.io/ofsoftswitch13/. [Online Accessed 2014-09-30].

[5] Flowgrammable. http://flowgrammable.org/. [Online Accessed: 2014-09-01].

[6] Manu Bansal, Jeffrey Mehlman, Sachin Katti, and Philip Levis. Openradio: a programmable wireless dataplane. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 109–114. ACM, 2012.

[7] Manu Bansal, Aaron Schulman, and Sachin Katti. Atomix: A framework for deploying signal processing applications on wireless infrastructure. In *NSDI*, pages 173–188, 2015.

[8] C Jasson Casey, Andrew Sutton, and Alex Sprintson. tinynbi: Distilling an api from essential openflow abstractions. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 37–42. ACM, 2014.

[9] Hsin-Hung Cho, Chin-Feng Lai, Timothy K Shih, and Han-Chieh Chao. Integration of sdr and sdn for 5g. *IEEE Access*, 2:1196–1204, 2014.

[10] Emrecan Demirors, George Sklivanitis, Tommaso Melodia, and Stella N Batalama. Rcube: Real-time reconfigurable radio framework with self-optimization capabilities. In *Sensing,*

Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on, pages 28–36. IEEE, 2015.

[11] Markus Fiedler and Tobias Hoßfeld. Quality of experience-related differential equations and provisioning-delivery hysteresis. In *21st ITC Specialist Seminar on Multimedia Applications-Traffic, Performance and QoE*. IEICE, 2010.

[12] Aditya Gudipati, Daniel Perry, Li Erran Li, and Sachin Katti. Softran: Software defined radio access network. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 25–30. ACM, 2013.

[13] Rohit Gupta, Bjoern Bachmann, Andreas Kruppe, Russell Ford, Sundeep Rangan, Nikhil Kundargi, Amal Ekbal, Karamvir Rathi, Arash Asadi, Vincenzo Mancuso, et al. Labview based software-defined physical/mac layer architecture for prototyping dense lte networks. 2015.

[14] Carlos Ramirez-Perez and Victor Ramos. Sdn meets sdr in self-organizing networks: fitting the pieces of network management. *IEEE Communications Magazine*, 54(1):48–57, 2016.

[15] ITUT Rec. P. 10/g. 100 (incl. amendment 2),âĂIJ. *Vocabulary for performance and quality of service*, 2008.

[16] ITU-T Recommendation. Perceptual evaluation of speech quality (pesq): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs. *Rec. ITU-T P. 862*, 2001.

[17] Roberto Riggio, Mahesh K Marina, Julius Schulz-Zander, Slawomir Kuklinski, Tinku Rasheed, et al. Programming abstractions for software-defined wireless networks. *IEEE Trans. Network and Service Management*, 12(2):146–162, 2015.

[18] Prithviraj Shome, Muxi Yan, Sayedjalil Modares Najafabad, Nicholas Mastronarde, and Alex Sprintson. Crossflow: A cross-layer architecture for sdr using sdn principles. In *Network*

Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on, pages 37–39. IEEE, 2015.

[19] Muxi Yan, Jasson Casey, Prithviraj Shome, Alex Sprintson, and Andrew Sutton. Aetherflow: Principled wireless support in sdn. *arXiv preprint arXiv:1509.04745*, 2015.

[20] Kok-Kiong Yap, Masayoshi Kobayashi, Rob Sherwood, Te-Yuan Huang, Michael Chan, Nikhil Handigol, and Nick McKeown. Openroads: Empowering research in mobile networks. *ACM SIGCOMM Computer Communication Review*, 40(1):125–126, 2010.

APPENDIX A

MODULATION SCHEMES

There are multiple ways modulating the periodic signals to represent digital binary data. Phase, frequency and amplitude are major parameters of signals that is varied to represent binary data. We use phase modulation in our experiment and will discuss it briefly.

A.1 Phase Shift Keying

The phase of the carrier is varied to denote the digital symbols. Different types are PSK modulations scheme are defined according to the number of bits per symbol the carrier signals carry.

We mainly consider PSK modulation techniques in our work. Some benefits of PSK modulation techniques are:

- Phase shift keying is more resilient to additive white gaussian noise than amplitude shift keying. The bit error rate for ASK is greater than PSK. PSK modulated signals do not carry information in the amplitude, whereas ASK do and the noise affects the amplitude of the signal which corrupts the ASK modulated data more than the PSK modulated data.
- For the same capacity and bit error rate, Phase shift keying has a smaller bandwidth than frequency shift keying schemes.
- In PSK, information of the transmitted signal is stored in phase variations which is more resilient to noise compared to amplitude or frequency variations.

A.1.1 Binary Phase Shift Keying

Binary phase shift Keying carries one bit per symbol. Two waveforms carrying 0 and 1 bit have same frequency and amplitude but are separated by 180 degrees making it antipodal in nature. The error rate for BPSK is the lowest compared to all the other PSK. This can be accounted for the maximum distance between the constellation point for 0 and constellation point of 1. This reduces the error in prediction when the received constellation points are skewed.

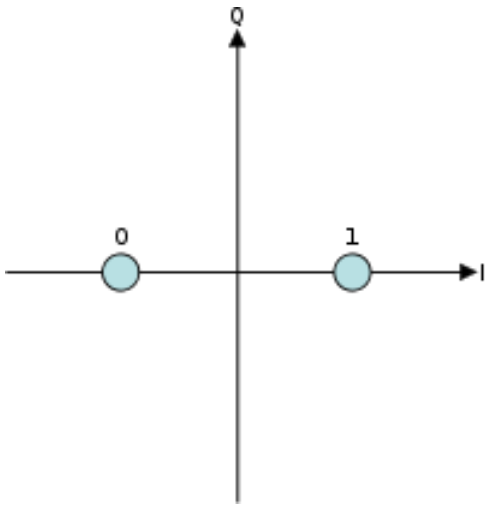

Figure A.1: Constellation plot for BPSK

Figure A.1 illustrates the constellation points for BPSK modulated data.

A.1.2 Quadrature Phase Shift Keying

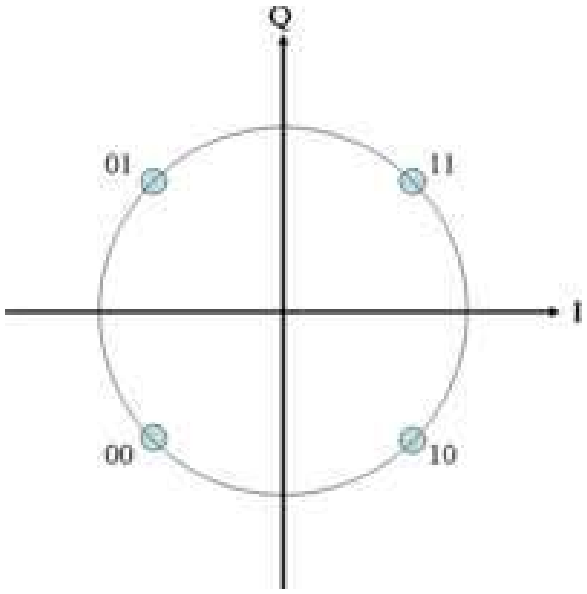

Figure A.2: Constellation plot for QPSK

Figure A.2 depicts the constellation points for QPSK modulated data.

Quadrature phase shift Keying carries two bits per symbol. The phase difference is 90 degrees for four different symbols. Data is modulated in quadrature and in-phase carriers. These two carriers do not interfere with each other due to the 90 degree offset at the receiver end. QPSK have a higher data rate, almost twice that of BPSK as each symbol transmit two bits. However, the reliability is not as high as BPSK.